Introduction to QuickCalc BASIC

Note: Please read the User License Agreement, below.

QuickCalc is a BASIC language interpreter. BASIC was developed a long time ago when PCs were brand-new and had very little (kilobytes) of memory and ran very slowly. Only a few people could program them so a simple language (BASIC) was developed to allow everyone else to use computers to solve problems and do math. Today, computers, even the cheap ones, run at processor speeds in the gigahertz range and have gigabytes of memory, and thousands of applications are available to make your computer do anything you want.

Sometimes, we just want to solve scientific problems and do quick math without a lot of programming knowledge. BASIC allows you to do that, and QuickCalc brings the functionality and ease-of-use of BASIC into the modern computer age.

In addition, QuickCalc introduces **high-speed, high-precision (long number) arithmetic and functions**, which are as easy to use as BASIC itself.

QuickCalc BASIC provides support for working with **Dates and Times**, by introducing a DATETIME type variable, which handles time values from microseconds to billions of years. These values are formatted correctly, taking into account leap-years, daylight saving time, etc., and can be displayed on graphs (See "Working with Dates and Times").

QUICKCALC Sample Programs.

Go to www.quickcalcbasic.com/ samples to download sample programs to run in QuickCalc BASIC. This collection of programs is designed to illustrate what you can do using QuickCalc. You can, and are invited to submit your own favorite programs for others to share.

Note: The Sample Programs are, by default, installed into "*My Documents*" \ QuickCalc.

QUICKCALC BASIC *vs* IBM BASIC

QuickCalc is patterned after the BASIC which was used on the original IBM Personal Computer. That computer used only DOS (Windows hadn't been invented yet), and therefore was built around things like text-mode graphics, screen formatting, communications protocol, etc. Today, Windows handles most of that for us, so QuickCalc concentrates on the programming and math.

Number Formats.

There is no integer variable type or single-precision floating-point. All numbers are **DOUBLE** floating-point. They work just as well, although not as fast. Nothing is lost storing integers as DOUBLEs – all significant digits are stored. Speed is not a major requirement here, nor is space. BASIC was originally developed to work on early IBM PCs and Apple-IIs. We have gigabyte computers and are running under Windows.

DATETIME variables are just another type of floating-point variable.

In addition, QuickCalc introduces **long decimal numbers** (**LONGMATH**) which may be as long as you want (up to one million digits). LONGMATH numbers may, in most cases be used anywhere regular (DOUBLE) numbers are used, and may be combined with them. All arithmetic and trigonometric functions work on long numbers as well. (see the sections on "Number Representation and Conversion" and "Using Long Numbers").

Graphic Functions

QuickCalc now provides a simple method for **graphing your data**. Graphs are produced with just a few statements and can be viewed on the screen and printed, or a graphic image file (Windows Enhanced Metafile or JPEG) can be created that can be used in other documents. Graphing is very much automatic – automatic scaling, axis generation, grid lines, etc. Most parameters can be overridden if desired. This version also supports logarithmic scales and DATE/TIME scales. Many shape drawing functions are also provided to create more interesting graphs.

Note: Graphing is not supported in CONSOLE mode.

See the help documents "*Beginning Graphics*", "*Intermediate Graphics*" and "*Advanced Graphics*" for a complete description of QuickCalc's graphing functions.

You can also write data to a text file directly with the **PRINT#** and **WRITE#** and use the file as input to a separate graphing program, such as GNUPLOT (www.gnuplot.info), for graphing functions that are beyond the scope of this program. You can create a metafile and import it into a graphic editing program to do more advanced formatting.

See the sample programs (www.quickcalc.com/samples) for many examples of how to create graphs.

Communications Functions

QuickCalc BASIC supports file I/O over serial ports, to facilitate communicating with microcontrollers (*e.g.*, Arduino) and serial peripherals (over serial or USB ports). It does not support remote terminals and modem communications functions (welcome to the 20th century). Just e-mail the file, or download it from the web

Editing

You don't edit from the command line anymore. Use **Notepad** to edit your program, then save it and run it in QuickCalc. It's a lot easier, and you can have both QuickCalc and Notepad open at the same time.

Error Handling

QuickCalc does not support error-handling procedures. It has a comprehensive list of error messages and checks for range and proper type on all parameters and operations, letting you know, with incredible detail, what went wrong and where. All errors will terminate the BASIC program, but QuickCalc itself should remain alive and ready to re-run the program once you have fixed the program or data.

Beyond that, it is your responsibility to check the range of your data values if you want to assure that the program will run smoothly

Loading images or Assembly programs.

Loading images or assembly programs is not supported, although you can launch other programs with the **SPAWN** statement..

Notes on Programming in QuickCalc BASIC.

Case is Ignored

abc and ABC and aBC all reference the same variable or reserved word. Throughout this documentation, BASIC functions, statements, commands and other reserved words are shown in **SMALL BOLD CAPITALS** for clarity, however upper or lower case is fine in your programs.

Blanks are ignored

except inside strings and where required for syntax. A=B+C is the same as A = B + C.

Tab characters are treated like spaces

You may use tab characters anywhere (except inside constants) to make the program more readable. Be aware that tab characters may not print the way you expect:

Tab characters (hex '09') that appear inside string constants will be handled like any other character. When printing them to the console, tabs will cause the console "printer" to skip to the next standard tab position for the console (1 plus a multiple of 8, *i.e.*, 1, 9, 17, 25, 33,…), ***not*** the value in PRINTTABS.. Listing a program with tabs may not format correctly, as there may be an offset from the margin when listing. Printing directly from Notepad will usually work, depending on your printer.

// (comment)

// may be used in place of **REM** to indicate that the rest of the line is a comment (similar to C-language).

Variable and array names

Names must start with a letter.
Names may contain letters, numbers, and $.
(new): Names may also contain the **underscore** (_) character (similar to C-language).
Names ending in $ are considered string variables.
Names cannot be the same as any command, statement, function or operator (see "*Functions, Statements and Commands*"). If you want to see a list of all the "reserved words" that cannot be used as variable names, enter the command **PRINTALL RESERVED**.
Maximum of 60 characters.
A name is unique if it differs from another in any position (except for case).
An array may not have the same name as a scalar or "non-array" variable (i.e., you cannot have a variable *abc* and an array *abc* (*m*, *n*) in the same program.)

Indirect Name Reference (*advanced programming function*)

A variable or array name may be specified *at execution time* as the contents of a string. Use the function @(*string-expression*). The string expression must contain the name of a variable or array. The resulting variable will be used in the statement. Example:

a = 45
s$ = "a"
b = @(*s*$)

The indirect reference @(*s*$) looks up the current value of *s*$ which is "a". The variable with the name *a* is then looked up. It contains the value 45, which is then assigned to *b*.

Indirect name references may be used anywhere a name is referenced, and can allow added flexibility in programming such as passing arrays into subroutines, etc. For more information, see the document "*Advanced Features of QuickCalc*".

Arrays

Maximum number of dimensions (order) = 6. Array subscripts are zero-based, *i.e.*, if an array is dimensioned to be *abc* (5) then it contains elements *abc* (0) through *abc* (4).

Note: A 6-dimensional array with 10 values for each dimension would have 10^6 or 1,000,000 entries. Don't make your arrays any larger than necessary.

Numbers

(This topic is discussed in detail in the section "Number Representation and Conversion")

Briefly, for short (DOUBLE) numbers:
Constants used in program may contain digits, signs, and decimal point, as appropriate.
They may have exponent (E+nnn or E-nnn).
Any invalid character terminates the number string, which may be no longer than 255 chars.

Numbers will be converted into internal floating point, which may shorten them or lose some of their precision.

For long (LONGMATH) numbers:

The same, except that exponents may go as high as 1,000,000.

Strings

Strings may be any sequence of characters (except "null") from 0 to 255 characters long. They may not carry over from one program line to the next.

Strings used in expressions are normally enclosed in quotes ("). Use (\") to insert a quote into the string. \\ inserts a single \ into the string. \r inserts a carriage-return, and \n inserts a line-feed, although it is not recommended to include c/r and l-f in strings. Number strings do not need quotes.

Strings used in **INPUT**, **DATA**, **INPUT#** do not need to be enclosed in quotes, unless they contain any of the following:

commas,
colons,
quotes,
significant leading or trailing blanks,
significant leading or trailing tab characters,
embedded escape sequences (\", \n or \r),
// (would be confused with a comment)

or

start with a digit, sign, or decimal point (*i.e.*, must not be confused with a number).

For all "funny" (non-keyboard) characters, use **CHR$**(*n*), *e.g.*,
a$="123"+**CHR$**(48)+"456".

An assignment statement for a string will re-use the same string storage if the new string length is less than or equal to the previous one. Any string operation that results in a string longer than 255 will be truncated to 255.

Built-in Constants

These may be used, but not changed.

PI=3.14159265358979334	constant π (same as 4 * **ATN** (1))
EULER= 2.7182818284590451	constant e (same as **EXP** (1))
DEFAULTPRINTFORMAT$$= "%13.6g"	default format for printing (see **PRINT** and **STR$**)
DEFAULTPRINTTABS=14	width of "tab" fields on screen (see **PRINT**)
DATE$="mm-dd-yyyy"	current date at time QuickCalc program starts

TIME$="hh:mm:ss"	current time at time QuickCalc program starts.
DAYOFWEEK$="ddd"	day of week at time QuickCalc program starts
FLOATINGPTLENGTH	current floating-point length (set by **FLOAT**)
ZERODATETIME	location of axis on DATETIME graphs (see "Working with Dates and Times")

Logical Expressions

Logical operators can operate on numbers and/or strings (mixed)

Zero = FALSE, non-zero = TRUE, null string = FALSE, non-null = TRUE.

Logical Expressions return FALSE = 0, TRUE = 1 (stored as numbers)
Order of precedence: comparison (=, >, <, <=, >=, <>) *[highest]*,
NOT,
Logical (**AND, OR, XOR, EOR)** *[lowest]*.

Arithmetic expressions

Arithmetic expressions are evaluated to yield a single number.

Order of precedence: (anything enclosed in parentheses) *[highest]*,
Functions,
Unary + or -,
Raise to power (a **^** b),
Multiply (*), divide (/) or remainder (%),
Add and subtract,
Logical operators (see above) *[lowest]*.

Operations which are higher in the order of precedence are performed first.

Statement Numbers

Must be integers and be >0 and < 2147483647.
Must start in column 1 of the line.
No duplicates.
Statement numbers do not have to be in sequence within the program.
Statement numbers are Optional. Only required for the target of **GOTO** and **GOSUB** and with a **DATA** statement referenced by **RESTORE** .
Statement numbers are checked to prevent branching into or out of a **WHILE** or **FOR** loop.

Multiple statements on one line:

Statements may be placed on the same line, up to the maximum line length of 255.
However, programs are more readable if statements are kept short, around 80 characters.
In many cases, a colon is required between statements.
If a statement ends with an expression (*e.g.*, *a* = *b*+2), the expression is automatically terminated by another statement that begins with a reserved word (*e.g.*, **PRINT** x).
However, if the next statement begins with a variable name or number (*e.g.*, *c*=**SQR**(2)) or a statement which could also be a function (like **INPUT**), you would need a colon to separate the two statements. It never hurts to use the colon for clarity.

A **REM** (or //) statement ends the line; all further characters are ignored.

IF / THEN [/**ELSE**] statements MUST be contained on a single line. They may be nested.
Two successive **ELSE** keywords must be separated by a colon. (see "Functions, Statements and Commands" for more notes concerning **IF/THEN/ELSE**).

All other statements may be placed together or on separate lines, whatever looks the best. Statements may not continue across multiple lines.

Note: Notepad will wrap long lines, making it appear as if they were multiple lines, which can be confusing. This is a good reason for keeping lines short.

Note: A command entered on the command line may not be followed by another statement on the same line.

Number formatting (in PRINT, PRINT USING, WRITE and STR$)

The default formatting for **PRINT** and **STR$** is given in ***defaultprintformat$$***="%13.6g".
Write statements are formatted with "%22.16g".
The value, if not null, in ***printformat$$*** (which you can change) overrides this.
PRINT USING specifies a print format, which is good for that statement only.
STR$ has an optional second parameter, which is an over-riding format string for that statement.
All of these statements (except **WRITE**) use the same type of formatting specifications:

C-type formats (see C-language specs for sprintf()).
Look on http://msdn.microsoft.com/en-us/library/56e442dc(VS.71).aspx.
Starts with %.
Must specify a single numeric variable.
Must be a valid c-type format spec. Could crash program is invalid.
"%23.18g" will show the most precision in a floating-point number (see "max").

Basic-type formats *(some differences from IBM BASIC here)*:

"#,###,###.##"

represents a digit, sign, or leading blank.
Commas are optional, only printed if a digit is left of them.
+ in first position means print a + or – sign.
(sign after is not implemented)
*(leading ** not implemented)*
($$ not implemented)
(^^^^ not mplemented)
Any number too large will be printed in exponential format.
(__ literals not implemented)
"###.cc" takes "integer" cents and formats it as dollars.cents.*(new)*
Maximum print width is 24.

"MAX":

You can specify "max" (in quotes) as the format string in **PRINT USING** or as the second parameter in **STR$** ( ). This forces the format string to be "%23.18g" which will show the maximum precision in a DOUBLE floating-point number. You can also set **PRINTFORMAT$$=**"max" to make it the default.

"ENG*xxx*"

This specifies "Engineering Notation", useful in electronics.
xxx is a string representing the units, and will be printed along with the number, *e.g.*, 25.4 m-amp where *xxx* is "amp".
The number is formatted as a 3-digit number with a decimal point, where the answer is expressed in kilo-, micro, nano-, etc. Example: 23456 ohms is formatted as 23.5 K-ohm (rounded to 3 digits). There will always be 1, 2 or 3 digits left of the decimal point. The decimal points are kept aligned, regardless of the number of digits. The result is a string 13 characters long (including leading sign or space), which will fit in the default tabs width of 14, unless *xxx* is longer than 3 characters.
Numbers greater than 999e+014 or less than 1e-012 will be printed in exponent notation (e.g., -1.23e+015amp).
Note: This won't work on number strings, only variables or expressions, since number strings are printed like regular strings.
PRINT USING "eng*xxx*"; 12345+0 works,
as does **PRINT USING** "eng*xxx*"; **SHORT** (12345).

LONGMATH formatting. LONGMATH values may now be formatted with the **PRINT USING** statement (or the **STR$** function). The format is "*www*[.[*ppp*]][c]". (See "Working With Long Numbers".)

Note: Many special formats not implemented in the BASIC-type can be done using the C-type format spec. Also, you can concatenate strings to produce exactly what you want.

You can mix formats on a line using multiple print statements ending in semicolons. Example:

```
PRINT USING "###.##"; x;          // note ending semicolon
PRINT USING "ENGamp"; y           // continues on same line
```

Customizing QuickCalc

One of the goals in developing QuickCalc BASIC was to remove arbitrary limitations on number sizes, sizes of arrays, number of strings, etc. Some limits had to be chosen in order to build tables, etc., however most of those sizes can be changed using the "Customize" dialog.

Click the **Customize** button to open this dialog and change any values you desire. The program will keep you within safe limits. There is a help document available from inside the Customize dialog, describing each one of your options.

User License Agreement

This product (QuickCalc BASIC version 2.5) is provided as Freeware. It is not public domain software.

There is no charge to use it as often or as long as you like on as many computers as you like for any purpose. You may print copies of the documentation for your own use. You may redistribute it (give it away) only in its original compressed form (.quickcalcsetup.exe file). Note that the "freeware" designation may not apply to future versions of this product.

Freeware still costs money to produce. If you find this product useful, a donation to support the development of products like this would be appreciated. **A donation of $10 is suggested.** Please go to the **Help menu** and **click on the "CONTRIBUTE" button**.

This is copyrighted material. Your license does **not** give you the right to modify it, sell it or repackage it. You may not disassemble or reverse engineer it, or include portions of it in your own software without express written permission from the author.

Any reports, tables, or graphs produced from BASIC programs that you write are your property. Any such programs submitted for inclusion on the Sample Programs page become public domain.

This software is provided "as-is," without any express or implied warranty. We cannot and do not assume any liability for losses resulting from the use or misuse of this product, nor can we guarantee the accuracy of calculations. . In no event shall the author be held liable for any such damages. It is your responsibility to determine the suitability of this product to your application.

We welcome your feedback. Please send comments, problem reports, and suggestions for new features or improvements to feedback@QuickCalcBasic.com.

Acknowledgements:

The installer program, Inno Setup 5, was written by and used with permission from Jordan Russell. http://www.jrsoftware,org

The JPEG graphics creation portion of this software is based in part on the work of the Independent JPEG Group, and used with their permission. http://www.ijg.org/